Project Introduction

For this project, you will be building Travel Journey, a trip journal app that lets users share their travels with their friends and family. In this project, you'll build a comprehensive digital travel journal. Users of your app will be able to create accounts, log in, and document their travel experiences with detailed trip entries, photo uploads, and event recordings.

Project Summary

You'll start by setting up and running a local API on your localhost - this will be the backbone of your app. Through this, you'll learn how to handle user authentication, manage trip data, and work with media uploads.

But that's not all. To enrich the user experience, you'll integrate Apple's MapKit, allowing users to search for locations and mark their travels on interactive maps. This project is not just about coding; it's about creating an engaging and functional travel journal app, complete with user interactions and real-world data handling.

Your starting point is a Travel Journey app equipped with mock data. Your mission is to bring this app to life by integrating it with a live API running locally. This transformation involves replacing mock data with real network calls, handling asynchronous operations, and ensuring a responsive and seamless user experience.

This project is not just about fetching and displaying data; it's a journey into the intricacies of network programming, understanding API calls, parsing JSON data, and managing concurrent operations in a modern iOS app.

Why am I building this?

Engaging with the Travel Journey project is more than just a coding exercise. It's a deep dive into two pillars of iOS app development: networking and concurrency. Let's break down why these skills are not just important, but crucial:

- **Networking**: In today's connected world, an app's ability to interact with the internet is often its lifeblood. In this project, you'll go beyond the basics to understand how to efficiently make network requests, process responses, and handle data parsing. These skills are key to unlocking the full potential of any modern app.
- **Concurrency**: Users demand apps that are not only feature-rich but also lightning-fast and ultra-responsive. Here, you'll learn to manage multiple tasks simultaneously without compromising the user interface's smoothness. We'll explore how to perform background operations, update the UI without hitches, and handle complex user interactions without a glitch.
- **Real-World Application**: What you'll learn here isn't just theoretical; it's directly applicable to the demands of today's app development industry. The techniques and

patterns you'll master are the same ones used in commercial software and high-end applications.

By the time you complete the Travel Journey app, you'll have a strong grasp of networking and concurrency. This experience will pave the way for you to create high-performing iOS apps, equipped to handle the complexities of modern app development.

What are the Requirements?

To ensure the success and functionality of the TravelJournal app, the following requirements must be met:

- **Networking Integration**:
  - Implement and utilize the `JournalService+Live.swift` for real networking operations using `URLSession` and Swift's `async/await`.
  - Successfully replace mock data with live data fetched from the local API.
- **Model Compliance**:
  - All models in the `Models` folder should conform to `Encodable`, `Decodable`, or `Codable` for efficient JSON encoding and decoding.
- **Mock Implementation Removal**:
  - Delete the mock implementation file `JournalService+Mock.swift` to focus on the live networking logic.
- **Concurrency Management**:
  - Ensure that all asynchronous operations are managed correctly using Swift's concurrency features, maintaining app responsiveness and preventing UI freezes.
- **Interactive Features**:
  - (Optional) Add a feature to share trip details using Swift's `Transferable` protocol, enhancing the app’s interactivity and user engagement.

Meeting these requirements will not only demonstrate a solid understanding of networking and concurrency in Swift but also provide a strong foundation in building responsive and user-friendly iOS applications.

Project Instructions

Download the Project Files
In the resources section of the project lesson are two zip files: Travel Journey API.zip which contains the code required for the local application server and Travel Journey Starter.zip which contains the iOS app starter files. Download and extract the project files to two different directories on your computer.

Setting Up the Local API
Before diving into the app development, let's set up the foundation. You'll start by launching a local API on your localhost. This API will serve as the backend for the Travel Journey app, handling tasks like user authentication, trip data management, and media storage.

Steps to Get Started:

1. Navigate to the API project folder provided with the course materials.
2. Follow the instructions detailed in the API's README.md file to set up and run the API on your local machine. This may involve installing certain dependencies and starting the server.

Remember, this local API is a critical component of your project. It's not only about getting it up and running but also understanding how it interacts with the Travel Journey app. If you encounter any issues or have questions, the README.md is your go-to guide, packed with essential information and troubleshooting tips.

Starting with the Starter Project
Now, with your local API up and humming, begin by exploring the provided starter project. It contains a basic structure and mock data simulating travel journal entries. Familiarize yourself with its components, understanding how data flows within the app.

Key Components of the Starter Project
The starter project for the Travel Journey app includes several core components, each playing a crucial role in the app's architecture. Here's a closer look at what you'll be working with:

- **JournalService/JournalService.swift**:
  - JournalService.swift is a protocol that outlines the blueprint for networking operations within the app. Currently, it includes a mock implementation in *JournalService/JournalService+Mock.swift*, designed to simulate network interactions. Your challenge is to replace this mock version with a real implementation that performs actual networking tasks, including API calls and response handling.
- **Models/Models.swift**:

  - This file contains the data models that represent the structure of the API responses. These models are vital for decoding JSON data from the API into Swift objects. You'll find models here for token, trip, event, location, and other relevant data entities.
- **Models/Requests.swift**:
  - Here, you'll see the request structures for creating and updating data via the API. These request models determine how data is sent from the app to the API.

Your deep dive into these components is essential for the successful integration of the local API, transforming the Travel Journey app from a mock-up to a fully functional application.

Replacing Mock Data with Network Calls
Your primary task is to replace the mock data with actual data retrieved from a local API. Set up a local server to simulate an API if one isn't already provided.

Implementing Networking in Swift

- **Model Protocols**:
  - Review your models in `Models/Models.swift` to ensure they conform to the appropriate protocols for JSON encoding and decoding. This typically means making them conform to `Encodable`, `Decodable`, or `Codable` protocols. These protocols are fundamental for translating between JSON data and Swift objects, a critical process in networking.
- **Creating a Live Networking Layer**:
  - In a new file named `JournalService+Live.swift`, create a live implementation of the `JournalService` protocol. This is where you will replace the mock logic with actual networking code. Utilize `URLSession` and leverage Swift's modern `async/await` syntax for handling asynchronous network requests. Implementing the methods of `JournalService` in this file will be your key step in enabling real data communication between the app and the API.
- **Deleting Mock Service**:
  - As part of transitioning to live network interactions, locate and remove the mock implementation file `JournalService+Mock.swift`.

Before you submit, check your project deliverables against the project rubric.

Reminder: **Your app will NOT be sent or shared with Apple for review.** You are submitting it to project mentors and graders.

App Functionality

Criteria	Submission Requirements
Model Protocol Conformity	All models in the Models folder must conform to Encodable, Decodable, or Codable protocols.
Networking Implementation	Implement the app to successfully interact with the local API using JournalService+Live.swift and URLSession. Proper setup and teardown of network sessions are crucial. Use appropriate HTTP methods and headers, including tokens for each endpoint, as detailed in the API readme.
URLRequest Creation	Create a function in JournalService+Live.swift for crafting URLRequests, to be reused across different network calls.
Mock Service Removal	Remove the mock service implementation (e.g., JournalService+Mock.swift) from the project.
Functional Consistency	Maintain functional parity with the starter project. All interactions possible with the mock service should be fully operational with the live service.
Persistence Verification	Ensure all actions performed in the app are persistently stored and reflected in the API data upon app relaunch.
Concurrency Handling	Manage asynchronous operations using async/await and ensure UI updates are on the main thread.
UI Accuracy	The UI should accurately reflect the data retrieved from the API.